

A Practical Software Fault Measurement and Estimation Framework

Allen P. Nikora
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA 91109-8099
Allen.P.Nikora@jpl.nasa.gov

John C. Munson
Computer Science Department
University of Idaho
Moscow, ID 83844-1010
jmunson@cs.uidaho.edu

Abstract

Over the past several years, researchers have been investigating methods of estimating the number of faults inserted into a software system during its development. One technique that has been developed is based on the observation that the amount of measured structural change between subsequent versions of a software system is strongly related to the number of faults inserted between those versions. If the structural evolution of a software system can be measured during the entire implementation activity, it is possible to estimate the number of total number of faults that have been inserted. Furthermore, the number of faults remaining when the system is turned over to operations can be estimated by subtracting the number of faults actually discovered and removed during testing and other fault identification activities (e.g., inspections) from the estimated total number of inserted faults. To make use of this technique, a practical structural measurement capability has to be developed and integrated into the software development environment. We describe a measurement capability we have implemented on several projects at the Jet Propulsion Laboratory, which consists of three components:

- Structural measurement –modules that have changed since the last set of measurements were taken are identified, measured against a baseline, and have their fault indices computed.
- Fault burden computation – the fault indices are used to compute each module's proportional or absolute fault burden.
- Fault measurement and identification – for each fault repaired, the point at which it was inserted into the system is determined. This information is used to develop a model from which absolute fault burdens can be estimated.

The first two components can be automated by integrating the measurement tools into the development environment. The third component remains a manual activity. However, even if the third component is not implemented, useful information in the form of proportional fault burdens at the module level may be obtained from the first two components.

We also identify issues associated with implementing a measurement framework. These include:

- Identifying a set of structural measurements.
- Creating standards for those measurements.
- Establishing a measurement baseline.
- Identifying and counting faults in a consistent manner.
- Discriminating between fault repair and other changes made to the software.

A Practical Software Fault Measurement and Estimation Framework

Allen P. Nikora

**Autonomy and Control Section
Jet Propulsion Laboratory
California Institute of
Technology
Allen.P.Nikora@jpl.nasa.gov**

John C. Munson

**Computer Science Department
University of Idaho
Moscow, ID 83844-1010
jmunson@cs.uidaho.edu**

The work described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The work was sponsored by the National Aeronautics and Space Administration's IV&V Facility and the JPL Center for Mission Information Systems and Software (CSMISS) Software Engineering Technology Work Area

Topics

- **Measurement Overview**
- **Fault Measurement and Estimation Framework**
 - ◆ **Structural Measurement**
 - ◆ **Fault Identification and Counting**
 - ◆ **Fault Burden Computation**
- **Summary**
- **References**

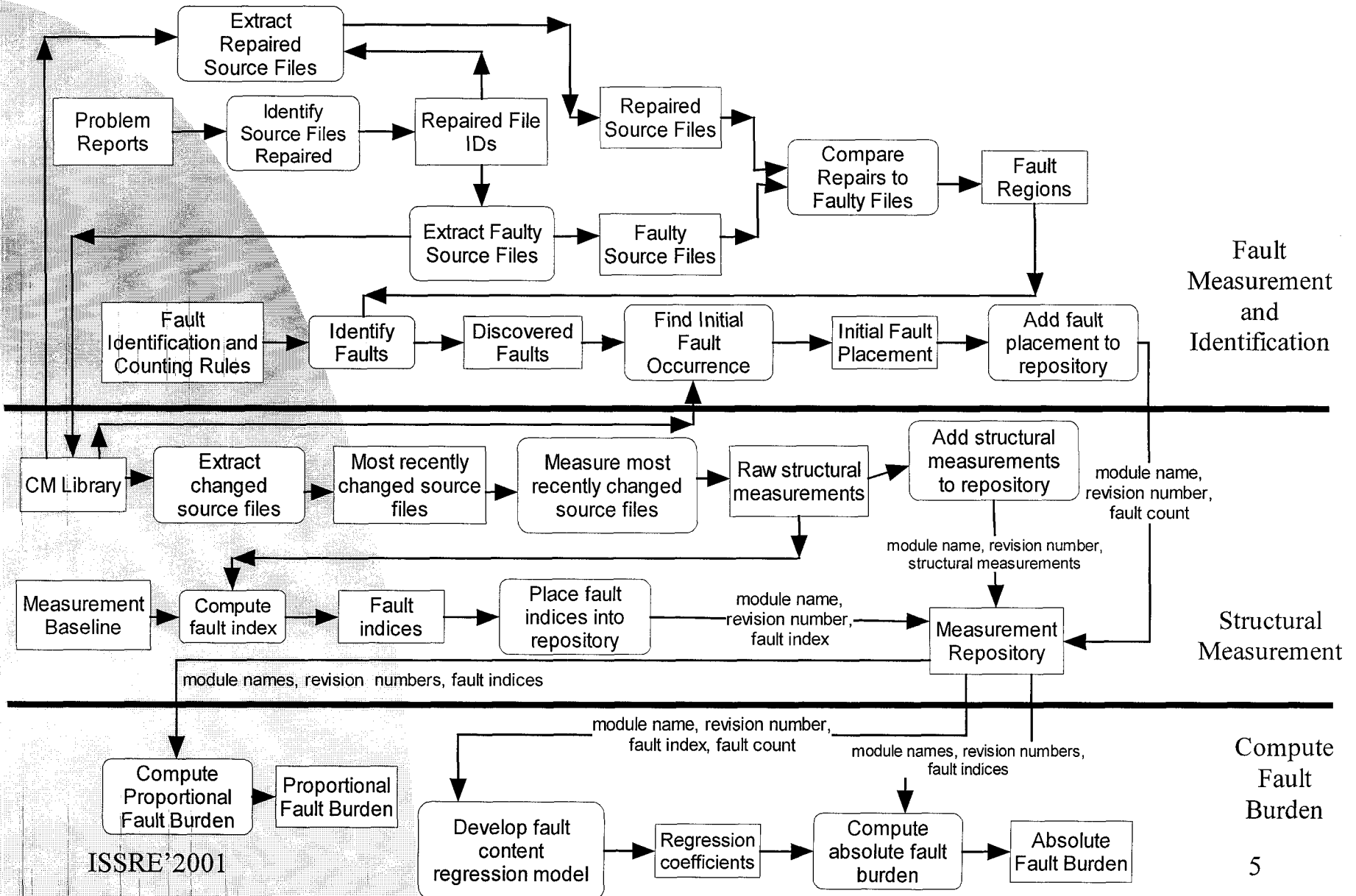
Measurement Overview

- **Measurement is central to any engineering process**
- **All software design decisions governed by measurable outcomes**
- **All code development controlled by measurable outcomes**
- **All software test activity controlled by measures of test activity**
- **All software reliability decisions quantified**

Measurement Overview

- Fault content during development can be estimated using measurements of system's structural evolution [Mun98,Niko981].
 - ◆ Make structural measurements each time a new version of a module is checked into CM repository.
 - ◆ Analyze raw measurements with respect to a baseline set of measurements
 - ◆ Compute *fault index* via principal components analysis [Dillon84]
 - ◆ Track structural evolution by recording differences between fault indices for subsequent versions of each module comprising the system.
 - ✦ Difference is termed *fault change*
 - ✦ Absolute value of difference is termed *net fault change*

Fault Measurement and Estimation Framework



Measurement Framework

- **Attributes**

- ♦ **Two of the three major areas can be automated**
- ♦ **Fault identification and counting is still a manual activity...**
 - ♦ **But is not necessary to get useful information.**
 - ♦ **Proportional fault burdens may be estimated without failure and fault information.**

Structural Measurement

- **Establish a measurement baseline**
 - ◆ **Performed infrequently**
 - ◆ **Must be done at least once at the start of a measurement effort**
 - ◆ **Baseline should also be changed as follows:**
 - ✦ **For a system with multiple releases (e.g., more than 2), change baseline after each release.**
 - ✦ **Re-establish baseline if measurement tools change – experience indicates that no two tools make measurements the same way**
 - ◆ **Establish separate baseline for each programming language**

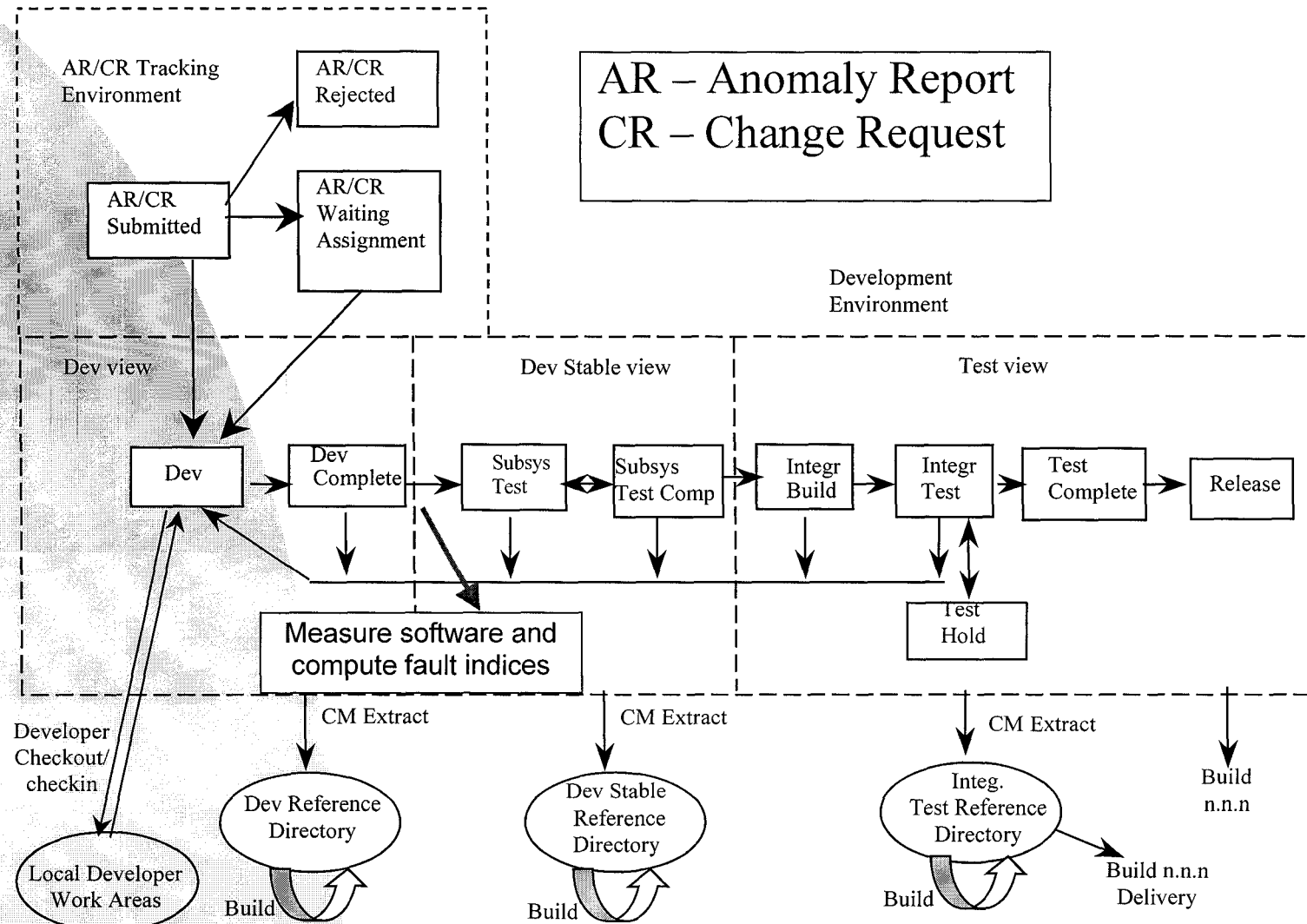
Structural Measurement (cont'd)

- **Identify source modules that have changed since the last set of structural measurements was made**
 - ◆ **Measure each time developers check modules into CM library**
 - ✦ **For a large development effort, measurement overhead might become large enough to pose burden on developers**
 - ◆ **At a regular time each day (preferably when few or no developers are on the system), identify all of the modules that have changed since the last time measurements of the system were taken**
 - ✦ **Developers don't notice measurement overhead**
 - ✦ **Failure of measurement mechanism will not affect developers**

Structural Measurement (cont'd)

- **Take structural measurements of the identified source modules**
 - ◆ **Measure components that are at the same level of maturity to ensure fault indices have the same meaning**
 - ◆ **CM systems may offer different views of a system**
 - ✦ components that have completed unit test
 - ✦ components that have completed integration test
 - ◆ **Need to strike balance**
 - ✦ compare systems of similar maturity, and
 - ✦ obtain enough measurements throughout the system's development to constitute a "good" change history
 - ✦ Recommend measuring components that have passed unit tests, but have not yet been integrated

Structural Measurement (cont'd)



Structural Measurement (cont'd)

- All measurement domains should be represented in measurement
 - ◆ Prior work indicates measurements fall into following domains [Mun98, Niko981]:
 - ✦ Size
 - ✦ Structure
 - ✦ Style
 - ✦ Nesting
 - ◆ Different languages may have different domains
 - ✦ Previous work did not include O-O measurement

Fault Index: the Fault Surrogate

- **Varies in direct proportion to software faults**
- **Used to estimate a module's fault burden (proportional or absolute)**
- **Ratio of two module's fault indices indicates how many more faults one module has than another**
- **Absolute fault burden can be estimated by using fault index as input to a (regression) model relating the fault index to the number of faults**

Fault Standard

- **A fault standard must be created for all fault recording processes.**
- **A valid fault standard has the properties that:**
 - ◆ **All developers will record faults in exactly the same way**
 - ◆ **All developers will enumerate faults in exactly the same manner.**
- **Fault standard must be evaluated by experiment**

Fault Identification and Counting

- **Identify failures and faults**
 - ◆ **For each failure, identify all source files changed in making repairs**
 - ◆ **Identify individual faults removed from software in response to a failure**
 - ✦ **Compare repaired source files to versions of those source files containing the faults**
 - ✦ **Apply fault identification and counting rules [Niko98,Niko981] to the resulting differences between the two sets of files.**
- **For each fault, identify point at which it was first inserted into the software**
 - ◆ **Search all previous versions of module to identify version in which fault first appeared**
 - ◆ **Measure structural difference between version in which fault initially appeared and immediately preceding version.**
 - ◆ **Develop regression model relating number of faults inserted per unit structural change.**

Fault Identification and Counting (cont'd)

- **Changes made in response to a failure must be separated from other types of changes (e.g., addition of new functionality, modification of existing functionality).**
 - ◆ **Failing to do so will lead to unquantifiable noise in the fault measurements**
 - ◆ **Accuracy of regression models will be reduced, and may make their construction impossible.**
- **Project management can establish and enforce following policy:**
 - ◆ **Do not make repairs and other types of changes to a component at the same time**
 - ◆ **First make the repairs, verify them, and check them into the configuration library**
 - ◆ **Use the repaired version as the basis for enhancing or adding functionality**

Fault Identification and Counting (cont'd)

- **Some revision control and problem reporting systems have features that will help to implement this policy. An example follows:**
 - ◆ **The CM system unit of work is a “change package”**
 - ✦ **Change packages created for problem reports, new functionality, or requirements change requests (CR)**
 - ✦ **Work associated with a problem report/CR/new functionality is checked into the change package.**
 - ◆ **Problem reporting system is tied to the CM system in such a way that for each new failure report created, a change package is created in the configuration library**
 - ✦ **Automatically provides a place for developers to submit the repairs**
 - ✦ **However, successful enforcement is ultimately the responsibility of project management and the development teams.**

Fault Burden Computation

- **Proportional Fault Burden**
 - ◆ **Compute ratio of its cumulative net fault change to the sum of the cumulative net fault change values for all modules in the system.**
 - ✦ **Example: if the cumulative net fault change for module A is 7, and the sum of the cumulative net fault change values for all modules in the system is 140, we would expect module A to have had inserted into it 7/140, or 5%, of the total number of faults inserted into the system. The proportional fault burden of module A would be 5%.**
- **Absolute Fault Burden**
 - ◆ **Use regression model to predict number of faults inserted**
 - ✦ **Inputs – fault change, net fault change**
 - ✦ **Output – number of faults inserted**

Summary

- Recent work showing that a software system's measured structural is related to its fault content has led to the development of a practical measurement framework.
- Structural measurement activities can be automated by means of scripts interacting with the revision control system being used for the development effort
 - ◆ Scripts run at regularly-scheduled intervals to
 - ✦ Identify and measure the modules that have changed since the last time measurements were taken
 - ✦ Compute fault indices
 - ◆ Fault indices alone can be used to estimate the proportional fault burden of a given module at any time.

Summary (cont'd)

- ◆ **Estimating absolute fault burden requires information obtained by tracing repaired faults back to the version of the module(s) in which they originally appeared**
 - ✦ **Cannot be completely automated**
 - ✦ **Also difficulty of separating changes due to fault repair from changes due to adding or enhancing functionality. Policies can be developed to require that these types of changes be given different types of labels in the configuration library, but software managers and development team leads are ultimately be responsible for enforcing such policies.**

References

[Dillon84]

W. R. Dillon, M. Goldstein, Multivariate Analysis Methods and Applications, John Wiley and Sons, 1984, ISBN 0-471-08317-8

[Mun98]

J. Munson, A. Nikora, "Estimating Rates of Fault Insertion and Test Effectiveness in Software Systems", proceedings of the Fourth ISSAT International Conference on Quality and Reliability in Design, Seattle, WA, August 12-14, 1998

[Niko98]

A. P. Nikora, "Software System Defect Content Prediction From Development Process and Product Characteristics", Ph.D. Dissertation, May 1998, University of Southern California, Computer Science Department

[Niko981]

A. Nikora, J. Munson, "Determining Fault Insertion Rates For Evolving Software Systems", proceedings of the Ninth International Symposium on Software Reliability Engineering, Paderborn, Germany, November 4-7, 1998.